Many (all) of my teams are adopting Docker for a full devops workflow and in doing so docker-compose is the de-facto standard for managing an application stack. However, the more sophisticated applications require configuration nuances between environments.

In AWS for example, one region may not have a service that is available in your other regions. Obviously you would try to avoid that scenario. However a far more common scenario is simply that your local environment will be connecting to other containers, while your staging and production environments will likely be communicating with "live" services.

In any case, the simplest way to manage this across a large team is by using `.env` files with your `docker-compose.yml` file so that developers can manage their environment without impacting the overall deployment definition.

Environment Variables

```
DB_ROOT_PASSWORD: r0ot!
DB_DATABASE: litwicki
DB_USER: username
DB_PASSWORD: password
DB_PORT: 3306
```

Now in your `docker-compose` you'll just reference this values.

```
db:
  image: "mysql"
  ports:
    - ${DB_PORT}
  environment:
    MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
    MYSQL_DATABASE: ${DB_NAME}
    MYSQL_USER: ${DB_USER}
    MYSQL_PASSWORD: ${DB_PASSWORD}
```

Repository Setup

Your repository should thus be setup with a standard `.env.dist` file that all developers can to their own, while your `.gitignore` naturally ignores the pure `.env` file so no real configuration values are ever stored in source control.

This simple example launches a *MySQL* container with a separate container for *adminer* as a web GUI to manage the database instance. Naturally you can expand this as you see fit and build your entire application stack with additional containers as needed, leveraging the environment configuration for the entire stack.

https://github.com/litwicki/docker-compose-example